L-4_Corp Embedded Linux

(c) kaiwanTECH. Click here to contact us.

Duration: 5 days	
Pre-requisites	
Mandatory	***Preferable***
Course L-0 “Linux CLI and Shell Scripting” *plus*	Strongly Recommended: Course level L-2 “Linux Kernel Internals”
Course C-1 “Programming in C” *plus*	
Course L-1 “Linux Systems Programming”	
Below is the Outline TOC (Table Of Contents) document: it presents the (approximate) Day-wise Coverage.	

Day 1

Module 1: Linux OS : Fundamental Prerequisites

- OS Architecture
  - Processor Protection Levels
  - Monolithic Kernel

Module 2: Embedded Linux Development

- Development ecosystem
  - Host and Target

GNU Toolchain

- Toolchain Components
- Building a Toolchain
  - The Roll-Your-Own Approach
  - Prebuilt Toolchains
  - Toolchain Build Utilities

Module 3 : The Linux Kernel Source Tree

- Who makes up the kernel dev community?
- Layout of the kernel source
  - A Brief Tour of the kernel source tree
- Kernel Releases
- Codebase Size

Module 4: Building a 2.6 / 3.x Linux™ Kernel

- The Kernel Repository
- The Kernel Development Process

Download and Extraction

Patches and Patch Management
Configuring the Kernel
Compiling the Kernel
Installing the Kernel modules

Day 2

Embedded Linux Emulation with QEMU

Module 5 : Embedding 2.6 / 3.x Linux™- A First Target: a small QEMU-based system

What is QEMU
Installation
Emulating an ARM Linux System
ARM kernel and disk image
Trying it out
User-Space Emulation with QEMU

Running a custom Linux kernel and Root Filesystem under QEMU

Mini-Project I : Embedding an ARM Linux kernel and (a simplistic) minimal root filesystem (no shared libraries) on an emulated e-Linux system.

Custom Scripts for building a QEMU/ARM eLinux system
Mini Project II : Embedding an ARM Linux kernel and minimal root filesystem (with shared libraries) on an emulated e-Linux system.

Day 3

Module 7 : Target Board Setup

Understanding the Board (the R Pi)
Communication between board and the host

Module 8 : Root Filesystem Content

BusyBox
Custom Applications
Libraries
System Initialization
Standard System V init
BusyBox init

Buildroot
Introduction
Configuration, uClibc-Buildroot Toolchain, Build
Root Filesystem Choices
Customizing

Module 9: Bootloaders

Introduction
u-boot : configuration, compilation & test cycle
Demo on an embedded-linux board / QEMU

Module 10 : Kernel Development – an Introduction

Practical Guide to Writing Kernel Code with LKMs
- Setting up your Test System
- The Hello World Module
- Compiling, the Makefile, Insertion and Removal
- Passing Parameters

Lab Assigments

Memory Management
- The VM Model
- Essential kernel MM APIs and their usage

Day 4

Module 11 : Kernel VFS and Open Files

Role of the kernel VFS Layer

The Open Files Table - the files_struct structure
The File Table - the file structure
The File Operations pointer - the file_operations structure

Kernel VFS filesystem I/O call graph

Module 12 : Character Device Drivers – the Framework

Introduction
Block and Character Drivers
Namespaces
- Major, Minor number
- Official Device Registry
- Inode Changes

User-Space vs Kernel-Space Drivers

Kernel-Space Character Drivers
The kernel “mem” driver
- The VFS layer and driver registration
- Copying User <--> Kernel Memory

Writing a character device driver for Linux - the Framework
- Registering with the kernel
- The major-wide open and fop setup
- Minor-specific open

Writing our own character driver: the zero and null memory devices
Device Nodes Management

Default Behaviour of f_op methods
Enhanced zero source functionality

Userspace Entry Points
A mention of
Procfs
Sysfs
Netlink
Debugfs

Accessing IO Memory
Accessing hardware registers and memory
ioremap, ioread, iowrite, etc
Examples
From kernel source tree drivers
Using the "Device IO Memory Read/Write" OSS project (time allowing, can demonstrate the same on an SBC)

Lab Assigment : use the above 'devmem' driver to examine kernel memory

Writing Reentrant-Safe driver code
Handling Blocking and Non-blocking I/O
Using mutexes for mutual exclusion control
Blocking I/O in practice – how to correctly handle read() / write() operations in char device drivers.

Day 5

Module 13 : Handling Hardware Interrupts

Interrupt Handling
do_IRQ()

Installing an Interrupt Handler
Interrupt Handler Flags
(Fairly) Recent Changes in Hardware Interrupt Handling in the Linux Kernel
Implementing an Interrupt Handler
Handler Arguments and Return Value
Threaded Interrupts

Interrupt Control
Disabling and Enabling Interrupts
Disabling a Specific Interrupt Line
Status of the Interrupt System

Deferred Functionality Mechanisms in the 2.6 Linux Kernel
- Tasklets and Bottom-Half Processing
- Softirq's
- A note on Work Queues
- Which One do I Use – A Quick Comparison

Module 14 : Linux Driver Model

The Sysfs filesystem
- Introduction
- Buses, Devices, Drivers
- Exploring

The Linux Driver Model
- Driver frameworks
- Bus Drivers
- Device Drivers – Unified Model
  - Defining
  - Registering
  - Probing and Removal hooks
- Platform Devices and Drivers

Linux and the Device Tree

Module 15 : Quick Overview of Kernel Debugging

Code-based techniques
- The printk, ratelimiting, MSG macros
- Convenience Header
- Tracing with Ftrace

Analysing an Oops dump
Tools, Probing
- A mention of KGDB and KDB

Kexec/Crash

Wrap Up.